Network Working Group
Request for Comments: 1730
Category: Standards Track

M. Crispin
University of Washington
December 1994

INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

The Internet Message Access Protocol, Version 4 (IMAP4) allows a client to access and manipulate electronic mail messages on a server. IMAP4 permits manipulation of remote message folders, called "mailboxes", in a way that is functionally equivalent to local mailboxes. IMAP4 also provides the capability for an offline client to resynchronize with the server (see also [IMAP-DISC]).

IMAP4 includes operations for creating, deleting, and renaming mailboxes; checking for new messages; permanently removing messages; setting and clearing flags; RFC 822 and MIME parsing; searching; and selective fetching of message attributes, texts, and portions thereof. Messages in IMAP4 are accessed by the use of numbers. These numbers are either message sequence numbers (relative position from 1 to the number of messages in the mailbox) or unique identifiers (immutable, strictly ascending values assigned to each message, but which are not necessarily contiguous).

IMAP4 supports a single server. A mechanism for supporting multiple IMAP4 servers is discussed in [IMSP].

IMAP4 does not specify a means of posting mail; this function is handled by a mail transfer protocol such as [SMTP].

IMAP4 is designed to be upwards compatible from the [IMAP2] protocol. Compatibility issues are discussed in [IMAP-COMPAT].

Table of Contents

IMAP4 Protocol Specification

1. Organization of this Document

1.1. How to Read This Document

This document is written from the point of view of the implementor of an IMAP4 client or server. Beyond the protocol overview in section 2, it is not optimized for someone trying to understand the operation of the protocol. The material in sections 3 through 5 provides the general context and definitions with which IMAP4 operates.

Sections 6, 7, and 9 describe the IMAP commands, responses, and syntax, respectively. The relationships among these are such that it is almost impossible to understand any of them separately. In particular, one should not attempt to deduce command syntax from the command section alone; one should instead refer to the formal syntax section.

1.2. Conventions Used in this Document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

2. Protocol Overview

2.1. Link Level

The IMAP4 protocol assumes a reliable data stream such as provided by TCP. When TCP is used, an IMAP4 server listens on port 143.

2.2. Commands and Responses

An IMAP4 session consists of the establishment of a client/server connection, an initial greeting from the server, and client/server interactions. These client/server interactions consist of a client command, server data, and a server completion result response.

All interactions transmitted by client and server are in the form of lines; that is, strings that end with a CRLF. The protocol receiver of an IMAP4 client or server is either reading a line, or is reading a sequence of octets with a known count followed by a line.

2.2.1. Client Protocol Sender and Server Protocol Receiver

The client command begins an operation. Each client command is prefixed with a identifier (typically a short alphanumeric string, e.g. A0001, A0002, etc.) called a "tag". A different tag is generated by the client for each command.

There are two cases in which a line from the client does not represent a complete command. In one case, a command argument is quoted with an octet count (see the description of literal in String under Data Formats); in the other case, the command arguments require server feedback (see the AUTHENTICATE command). In either case, the server sends a command continuation request response if it is ready for the octets (if appropriate) and the remainder of the command. This response is prefixed with the token "+".

> Note: If, instead, the server detected an error in the command, it sends a BAD completion response with tag matching the command (as described below) to reject the command and prevent the client from sending any more of the command.
>
> It is also possible for the server to send a completion response for some other command (if multiple commands are in progress), or untagged data. In either case, the command continuation request is still pending; the client takes the appropriate action for the response, and reads another response from the server.

The protocol receiver of an IMAP4 server reads a command line from the client, parses the command and its arguments, and transmits server data and a server command completion result response.

2.2.2. Server Protocol Sender and Client Protocol Receiver

Data transmitted by the server to the client and status responses that do not indicate command completion are prefixed with the token "*", and are called untagged responses.

Server data may be sent as a result of a client command, or may be sent unilaterally by the server. There is no syntactic difference between server data that resulted from a specific command and server data that were sent unilaterally.

The server completion result response indicates the success or failure of the operation. It is tagged with the same tag as the client command which began the operation. Thus, if more than one

command is in progress, the tag in a server completion response identifies the command to which the response applies. There are three possible server completion responses: OK (indicating success), NO (indicating failure), or BAD (indicating protocol error such as unrecognized command or command syntax error).

The protocol receiver of an IMAP4 client reads a response line from the server. It then takes action on the response based upon the first token of the response, which may be a tag, a "*", or a "+". As described above.

A client MUST be prepared to accept any server response at all times. This includes server data that it may not have requested. Server data SHOULD be recorded, so that the client can reference its recorded copy rather than sending a command to the server to request the data. In the case of certain server data, recording the data is mandatory.

This topic is discussed in greater detail in the Server Responses section.

of a message, and hence the minimum for the starting octet, is octet 1.

The following FETCH items are valid data for PARTIAL: RFC822, RFC822.HEADER, RFC822.TEXT, BODY[section], as well as any .PEEK forms of these.

Any partial fetch that attempts to read beyond the end of the text is truncated as appropriate. If the starting octet is beyond the end of the text, an empty string is returned.

The data are returned with the FETCH response. There is no indication of the range of the partial data in this response. It is not possible to stream multiple PARTIAL commands of the same data item without processing and synchronizing at each step, since streamed commands may be executed out of order.

There is no requirement that partial fetches follow any sequence. For example, if a partial fetch of octets 1 through 10000 breaks in an awkward place for BASE64 decoding, it is permitted to continue with a partial fetch of 9987 through 19987, etc.

The handling of the \Seen flag is the same as in the associated FETCH command.

```
Example:    C: A005 PARTIAL 4 RFC822 1 1024
            S: * 1 FETCH (RFC822 {1024}
            S: Return-Path: <gray@cac.washington.edu>
            S: ...
            S: ......... FLAGS (\Seen))
            S: A005 OK PARTIAL completed
```

6.4.7. STORE Command

```
Arguments:  message set
            message data item name
            value for message data item

Data:       untagged responses: FETCH

Result:     OK - store completed
            NO - store error: can't store that data
            BAD - command unknown or arguments invalid
```

The STORE command alters data associated with a message in the mailbox. Normally, STORE will return the updated value of the data with an untagged FETCH response. A suffix of ".SILENT" in

6.4.8. COPY Command

Arguments: message set
mailbox name

Data: no specific data for this command

Result: OK - copy completed
NO - copy error: can't copy those messages or to that name
BAD - command unknown or arguments invalid

The COPY command copies the specified message(s) to the specified destination mailbox. The flags and internal date of the message(s) SHOULD be preserved in the copy.

If the destination mailbox does not exist, a server SHOULD return an error. It SHOULD NOT automatically create the mailbox. Unless it is certain that the destination mailbox can not be created, the server MUST send the response code "[TRYCREATE]" as the prefix of the text of the tagged NO response. This gives a hint to the client that it can attempt a CREATE command and retry the COPY if the CREATE is successful.

If the COPY command is unsuccessful for any reason, server implementations MUST restore the destination mailbox to its state before the COPY attempt.

Example: C: A003 COPY 2:4 MEETING
S: A003 OK COPY completed

6.4.9. UID Command

Arguments: command name
command arguments

Data: untagged responses: FETCH, SEARCH

Result: OK - UID command completed
NO - UID command error
BAD - command unknown or arguments invalid

The UID command has two forms. In the first form, it takes as its arguments a COPY, FETCH, or STORE command with arguments appropriate for the associated command. However, the numbers in the message set argument are unique identifiers instead of message sequence numbers.

In the second form, the UID command takes a SEARCH command with SEARCH command arguments. The interpretation of the arguments is the same as with SEARCH; however, the numbers returned in a SEARCH response for a UID SEARCH command are unique identifiers instead of message sequence numbers. For example, the command UID SEARCH 1:100 UID 443:557 returns the unique identifiers corresponding to the intersection of the message sequence number set 1:100 and the UID set 443:557.

A unique identifier of a message is a number, and is guaranteed not to refer to any other message in the mailbox. Unique identifiers are assigned in a strictly ascending fashion for each message added to the mailbox. Unlike message sequence numbers, unique identifiers persist across sessions. This permits a client to resynchronize its state from a previous session with the server (e.g. disconnected or offline access clients); this is discussed further in [IMAP-DISC].

Associated with every mailbox is a unique identifier validity value, which is sent in an UIDVALIDITY response code in an OK untagged response at message selection time. If unique identifiers from an earlier session fail to persist to this session, the unique identifier validity value MUST be greater than in the earlier session.

> Note: An example of a good value to use for the unique identifier validity value would be a 32-bit representation of the creation date/time of the mailbox. It is alright to use a constant such as 1, but only if it guaranteed that unique identifers will never be reused, even in the case of a mailbox being deleted and a new mailbox by the same name created at some future time.

Message set ranges are permitted; however, there is no guarantee that unique identifiers be contiguous. A non-existent unique identifier within a message set range is ignored without any error message generated.

The number after the "*" in an untagged FETCH response is always a message sequence number, not a unique identifier, even for a UID command response. However, server implementations MUST implicitly include the UID message data item as part of any FETCH response caused by a UID command, regardless of whether UID was specified as a message data item to the FETCH.

C. References

[IMAP-AUTH] Myers, J., "IMAP4 Authentication Mechanism", RFC 1731. Carnegie-Mellon University, December 1994.

[IMAP-COMPAT] Crispin, M. "IMAP4 Compatibility with IMAP2 and IMAP2bis", RFC 1732, University of Washington, December 1994.

[IMAP-DISC] Austein, R. "Synchronization Operations for Disconnected IMAP4 Clients", Work in Progress.

[IMAP-MODEL] Crispin, M. "Distributed Electronic Mail Models in IMAP4", RFC 1733, University of Washington, December 1994.

[IMAP-NAMING] Crispin, M. "Mailbox Naming Convention in IMAP4", Work in Progress.

[IMAP2] Crispin, M., "Interactive Mail Access Protocol - Version 2", RFC 1176, University of Washington, August 1990.

[IMSP] Myers, J. "IMSP -- Internet Message Support Protocol", Work in Progress.

[MIME-1] Borenstein, N., and Freed, N., "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 1521, Bellcore, Innosoft, September 1993.

[MIME-2] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text", RFC 1522, University of Tennessee, September 1993.

[RFC-822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, University of Delaware, August 1982.

[SMTP] Postel, Jonathan B. "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.